Low-Latency Privacy-Enabled Context Distribution Architecture

João M. Gonçalves
Portugal Telecom Inovação S.A.
Aveiro, Portugal
Email: joao-m-goncalves@ptinovacao.pt

Diogo Gomes
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: dgomes@av.it.pt

Rui Aguiar
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: ruilaa@ua.pt

***Abstract*—As personal information and context sharing applications gain traction more attention is drawn to the associated privacy issues. These applications address privacy using an unsatisfactory "whitelist" approach, similar to social networks "friends". Some of them also link location publishing with user interaction which is also a form of privacy control - the user has to explicitly say where he is. There are a few automatic location based-services (LBS) that track the user, but without more adequate privacy protection mechanisms they enable even bigger threats to the user. On previous work, an XMPP-based Context Distribution Architecture was defined, more suitable for the distribution of frequently changing context than other systems because it is based on the publish-subscribe pattern. In this paper the authors present an extension to this architecture that allows for the introduction of a complex degree of access control in context distribution. The devised changes enable the system to consider a number of interesting context privacy settings for context distribution control. Also, this control must be enforced in a way that it doesn't interfere with the real-time nature of the distribution process. After describing the enhancements to the architecture, a prototype of the system is presented. Finally, the delivery latency and additional processing introduced by the access control components is estimated by testing it against the existing system.**

I. Introduction

There is still no comprehensive technological solution for addressing privacy in systems that make use of context information. An example of this are today's commercially available location sharing applications. In these applications access control is done by simply "whitelisting" [1] [2] users, enabling them to see the shared information. Furthermore, most of these applications require explicit user action for publishing information. The user has to say where he is from a location-aware list of places, an action commonly referred to as a "check-in". This is also a form of privacy protection since permanently tracking the user without further control is intrusive and uncommon [3]. However, in the future we will be sharing not only to people but also to the entities that administer the devices around us. Such entities will use this context information to adapt our physical and digital environment. This implies two things: first, the "check-in" technique currently used in most location sharing applications is not an option, and second, the latency performance requirements for the context distribution become tighter. The maximum delivery latency depends on the type and precision of the supplied context: more precise and dynamic context information requires less latency. In this work we assume that context information is only used for human-perceived adaptation. For this reason, the tightest latency requirements are bound by human reaction times. From the field of cognitive psychology, we consider as a reference the value of 190 milliseconds for human visual reaction time [4].

We built upon previous work with a practical approach to the problem, proving that it is possible to build such a privacy-enabled context delivery system today. The prototype and results presented in this paper show that it is possible to use complex privacy settings in context distribution without relevantly impacting the delivery latency. These complex privacy settings fit the user's privacy expectations better than the ones currently in use, enabling users concerned with privacy to share more [5] [2]. Furthermore, a context delivery system must have low context delivery latencies so that it can be used for adaptation scenarios. We believe that these two points are key enablers for the era of ambient intelligence.

The rest of the paper is organized as follows. In the next section we discuss related work on privacy-enabled context delivery. In Section 3 the architecture and design directives used to minimize the access control latency are discussed. In Section 4 the implementation details are presented. In Section 5 the performance results of the new prototype are matched against the pre-existing one. Finally, in Section 6 we conclude and present possibilities for future work.

II. Related Work

The work presented in this document builds upon work done in context management, context and personal information privacy and access control. In this section the each of the related work topics will be described, setting the ground for our enhancements.

In the field of context management, Abowd, Dey and Salber [6] propose a set of requirements for a context-enabling framework: separation of concerns, context interpretation, transparent and distributed communications, constant availability of context acquisition, context storage and history, and resource discovery. A number of context management architectures were presented over the last decade to address these problems

978-1-4577-2053-6/12/$31.00 ©2012 IEEE

[7]. A recent one put forward by Zafar et al. [8] generically addresses all the previously presented requirements. It successfully decouples the sources from the consumers, providing a discovery and communication platform between them by defining 4 main components: Context Source, Context Provider, Context Broker and Context Consumer. Furthermore it defines architectural points for the storage, history and inference functions. However, due to its request-response approach, it does not fully comply with the real-time requirements that adaptation scenarios have. Additionally, if the system is to be used globally, across trust domains, federation mechanisms must be in place. The architecture introduced by Gomes et al. [9] aims at solving these issues, while globally maintaining the architectural constructs. The work makes use of XMPP [10] as the main communication protocol because it responds to the mentioned requirements through the direct use of an existing standard.

Regarding context privacy work, location has enjoyed most attention. Toch et al. [11] and Benisch et al. [2] consider a location privacy preference model that uses not only whitelists but also the location information value, time of day and date. In their study, Benisch et al. measure the accuracy with which different privacy settings are able to capture the subject's preferences. Results show that using detailed privacy preferences such as date, time and location values leads to a 3 time increase in the accuracy of the settings, comparined to using whitelist-based settings only. Other location privacy work like the one by Ardagna et al. [12] use obfuscation techniques, which consist of deliberately providing less precise or even erroneous information. There is much done regarding the definition of specific obfuscation techniques for each context type. However, addressing context privacy generically has not been thoroughly explored. Wishart et al. [13] present a generic model for obfuscation, but different obfuscation ontologies have to be defined for each context type, since for each of them there are potentially different levels of detail that can be considered. Although we intend to tackle both access control and obfuscation approaches in our architecture, and to apply both to any context information type, so far only the first approach has been addressed. However, architecturally, we anticipate that generic context obfuscation will be made possible by the chaining of Context Providers. Regarding the first privacy approach, we managed to filter generically by context value with the use of a privacy settings format that dynamically references the context information parameters.

Most access control systems implemented today are proprietary and case specific. It was in order to bring some portability and standards to the area that Extensible Access Control Markup Language (XACML) [14] was developed. Version 2, released in 2004, is completely mature and very well known and version 3, released in 2010, is as of August 2011 in its first revision. It defines an architecture comprising of 4 components:

- Policy Enforcement Point (PEP) - entity that gets the request that needs to go through an access control check;
- Policy Decision Point (PDP) - central point of the architecture, receives XACML requests from the PEP and generates a decision, and subsequent response based on the existing policies;
- Policy Administration Point (PAP) - supplies access control policies to the PDP;
- Policy Information Point (PIP) - supplies relevant information to the PDP;

The main components are the PEP and the PDP. It is between them that the access control requests and responses flow. An XACML request represents a question: whether a given subject can do a given action to a specific resource. The PDP calculates the decision based on the policies it has. XACML defines an XML format for the policies. A policy refers to a target and contains a number of rules. Based on the request information, the PDP navigates the policies looking for a match in the target. When it does, it returns the associated decision, permit or deny. Note that policies can be nested, so combinatory algorithms may be used.

Work in the related areas is mature since context management is an area targeted for over 10 years, and access control has proven and widely adopted standards. Based on this work, and most relevantly on the context management architecture by Gomes et al. [9], we have built a prototype that implements a novel access control scheme inspired by the privacy work of Toch et al. [11].

III. Architecture

As described by Gomes et al. [9], context is distributed by publishing to certain "nodes" to which context consumers are subscribed. The nodes here play the role of resources: in a request-response case a request would be sent to a resource containing the targeted information. In traditional access control the access decision is done per request based on the requesting subject, the action being performed, the targeted resource, and environment parameters such as date and time [14]. However, in context distribution, in both for request-response and publish-subscribe cases, the resource is volatile: the values it contains change over time, in many cases rather quickly. Since we are aiming at considering the context values themselves for the decision, not only the resource identifier, the traditional access control model needs to be extended. In addition to this, the real-time nature of the context distribution process cannot be debilitated by the implemented access control mechanism.

Comparing to the original architecture, a component was introduced in the architecture, depicted in Figure 1, the Privacy Aggregator. This component is the user's contact point for choosing privacy settings, and it is necessary for two main reasons. First, communicating the privacy setting implications to the user is an important problem [5] that needs to be tackled with specific solutions [2]. Second, for the conception of the Privacy Aggregator relates to the concept and architectural definition of a Context Provider. Since Context Providers may belong to different entities, and a user typically interacts with more than one Context Provider, having one of them accessing the privacy preferences meant to another Provider

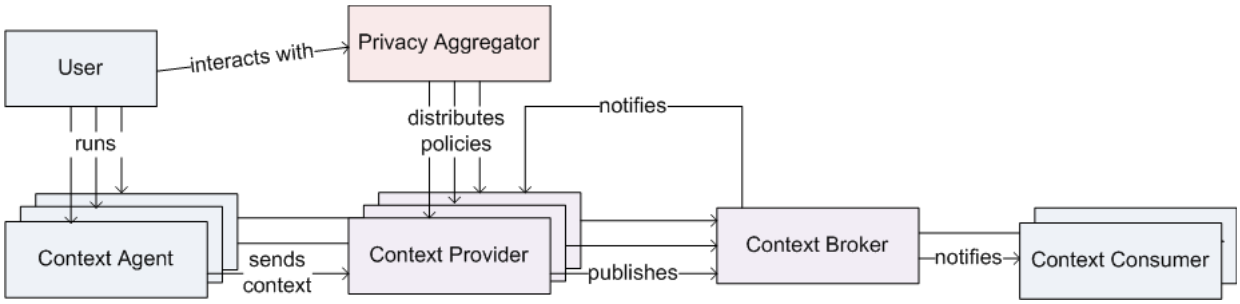

Fig. 1. Architecture with Privacy-related Access Control

is not acceptable. The Privacy Aggregator must work as a broker for specialized settings distribution. After the user sets his privacy options the Privacy Aggregator is responsible for separating them per provider and delivering them. Furthermore, the Privacy Aggregator is bound to perform a key role in chaining different Context Providers for obfuscating context. From the Context Providers' description of context inference and translation capabilities, the Privacy Aggregator can build relationships between the available context types. Some of these relationships will be obfuscations, and will replace the statically defined ontologies of previous work. For example, if a Context Provider exists that uses GPS location information to infer the city the user is in, then the Privacy Aggregator must associate this with the GPS Context Provider and take it in account when asking the user for privacy settings for the GPS context type. Finally, the question of what entity controls the Privacy Aggregator, whether by the user or by some entity that the user trusts (e.g., Identity Provider), is a subject for future work.

Not only is publish-subscribe better than request-response for context dissemination, due to the real-time requirements of context-awareness scenarios, it is also more flexible regarding access control. Request-response access control is done per request, and even if no environment parameters are considered - meaning that access decisions can be cached - there is always a check required per request. In publish-subscribe the access control can be enforced on subscription time, which happens typically a small fraction of the number times that context is required. Adopting his approach required changes in the original Context Broker.

The notion of "Context Profile" is introduced to ease the definition of the context privacy settings. A Context Profile refers to a context type and some optional privacy parameters: conditions expressing the values, dates and times for which the information can published, and an optional publishing delay. Then, in his privacy settings, the user associates each context profile to groups of entities that may be interested in taking the role of context consumers.

It is possible to define several Context Profiles for the same context type, even with privacy settings that are not mutually exclusive. Consequently the same context information may be distributed under different context profiles. Since the access control is done at subscription time, each context profile will necessarily correspond to a different publish-subscribe node, even if the same information is being published. While this brings an increase in the number of required nodes (or resources), and some replication of information, it will also reduce the context dissemination latency. It is a slight trade-off from horizontal scalability (the same server with the same processing power will typically support less users and load) to latency performance (context change notifications are as real-time as possible). By making this design choice, the only access control processing required at publishing time is the part of it that makes use of contextual information: context values, date and time of day.

The most relevant sub-components and interactions are depicted in Figure 2. The Privacy Aggregator provides the Context Profiles and associated users (or user groups) to the appropriate Context Providers, which implied changes in the original Context Provider. Since a Context Profile is context

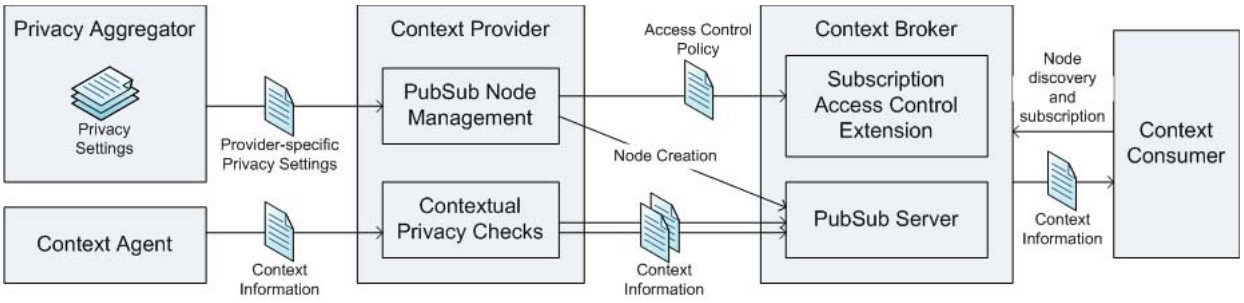

Fig. 2. Design specification based on cardinality relationships between context update, subscription and consumption

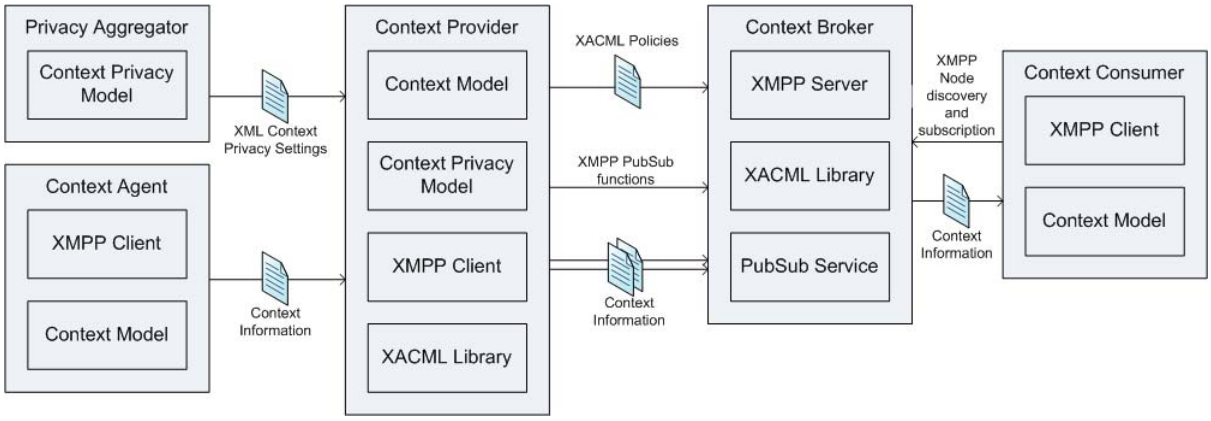

Fig. 3. Main implementation modules

type specific it must be supplied to the Providers that publish that type of context information, and only to those. The Context Provider is responsible for managing the publish-subscribe nodes it publishes to. It must create and destroy them, based on privacy configuration changes, and manage its associated access control policies - it plays the role of an XACML PAP. It also is responsible for the contextual checks required for discovering the appropriate nodes where to publish. This last process happens once per context update. The Context Broker only needs to enforce the defined node access control policies, both when discovering and subscribing to nodes. It plays both the PDP and PEP roles, although these may be decoupled.

IV. Implementation

For the prototype implementation we focused on the context distribution elements of the architecture because these are the ones that have an impact in the context delivery latency. The Privacy Aggregator was minimally implemented, only providing the specialized privacy settings XML to the Provider.

The components were implemented based on existing and new libraries. The implemented architectural components and main implementation modules are depicted in Figure 3. The Context Model is the module that allows parsing and understanding context information and it can be extended for different types of context. The Context Privacy Model is the module that allows parsing and understanding the defined context privacy settings.

The context privacy settings define context profiles, as explained before. In the Context Privacy Model implementation XML was used to serialize and transport the settings. The settings represent conditions to be matched to context values, date and time, in order to evaluate whether some context update is to be published under that profile or not. Although an extension mechanism was put in place for representing these conditions, in this implementation only regular expressions were used. An example of the preferences file is shown in Example 1.

In this example there are two GPS location profiles defined. The first one publishes location only in if the location is within certain boundaries, in this case nearby the city of Aveiro, where the user lives as depicted in Figure 4. The second profile publishes GPS location within a certain date/-time window: from 9:00AM to 19:59PM, from Monday to Friday. The first profile is made available to the user's home appliances (air conditioning, water heating, ...), here identified by "home@openfire", for them to be able to detect when the user is coming home. The second profile is made available to his work colleagues, here represented by every entity in the "openfire" domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<privacy user="jmgonc@openfire" xmlns="http://iex.ptin.pt/ctx/privacy
      ">
  <privacyProfiles>
    <profile id="1" name="GPS Tracking for Home Appliances">
      <publishedContextNamespace>http://iex.ptin.pt/ctx/gps</
           publishedContextNamespace>
      <parameterRules>
        <parameterRule>
          <parameter>latitude</parameter>
          <condition lang="regexp" var="">40\.6[2-5]\d*</condition>
        </parameterRule>
        <parameterRule>
          <parameter>longitude</parameter>
          <condition lang="regexp" var="">-8\.6[2-7]\d*</condition>
        </parameterRule>
      </parameterRules>
      <timeRule/>
      <publishingDelay>0</publishingDelay>
    </profile>
    <profile id="2" name="Friends Place Tracking">
      <publishedContextNamespace>http://iex.ptin.pt/ctx/gps</
           publishedContextNamespace>
      <parameterRules/>
      <timeRule>
        <timeOfDayCondition lang="regexp" var="">9|1\d</
             timeOfDayCondition>
        <weekdayCondition lang="regexp" var="">[2-5]</
             weekdayCondition>
        <dailyExceptionCondition lang="regexp" var="">0</
             dailyExceptionCondition>
      </timeRule>
      <publishingDelay>0</publishingDelay>
    </profile>
  </privacyProfiles>
  <accessGroups>
    <accessGroup id="1" name="Home Appliances">home@openfire</
         accessGroup>
    <accessGroup id="2" name="Work Colleagues">openfire</accessGroup>
  </accessGroups>
  <profileGrouplinks>
    <link groupId="1" profileId="1"/>
    <link groupId="2" profileId="2"/>
  </profileGrouplinks>
</privacy>
```

Example 1. Context Privacy Settings XML Representation

The Context Provider was coded using Java SE 6 and

Fig. 4. Area in which GPS Location is published for the first Context Profile

two existing Java open source libraries: Smack 3.2.0 [15] and Sun XACML 1.2 [16]. The provider is able to interpret the context privacy settings, and to create PubSub nodes and associated access policies in XACML which are supplied to the XMPP Server. The XACML policy only defines which users are allowed to perform a "read" action on a given resource (PubSub node). When context updates come in from Context Sources, the provider identifies which are the suitable nodes to which this information is published, first by checking the entity to which the context refers to, and then by enforcing the necessary parameter checks on that context information.

The Context Broker is implemented using Openfire [17], which defines a plugin architecture that makes it easy to enforce access control on node discovery and subscription. Such a plugin was coded, also using the open source Java library Sun XACML 1.2. The policy for each node is loaded by the plugin to a simple Policy Decision Point (PDP) implementation. Whenever a discovery request for PubSub nodes or a subscription request arrives, the user and resource information are passed to the PDP in order to get a decision on whether a resource is accessible or visible to that user.

V. Performance Evaluation

To test the system a testbed of virtual machines was setup. The host machine is an Intel Core i7 with 12Gb of RAM running VMWare ESXi 3.5. Four virtual machines were created with 512MB of RAM and a single virtual core, to ease the CPU Usage measurements. In all of them Ubuntu Linux 11.04 Server edition was installed. Each of the four machines runs one of the components: Context Source, Context Provider, Context Broker and Context Consumer. For the Context Source a simple Java-based XMPP client was coded, that generates random location context at a configured rate. Similarly, the Context Consumer is a Java-based XMPP client that subscribes to the target PubSub nodes and writes to a file the received context marked with a timestamp. The baseline Context Provider is a simple provider that publishes every context that receives to nodes that it has configured without any further processing. The Context Source and Consumer are synchronized automatically before each test using a local NTP server.

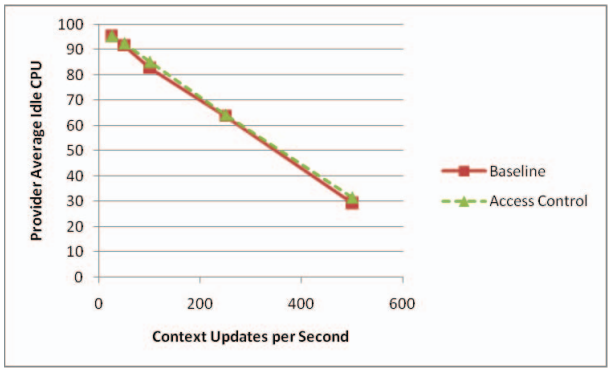

Fig. 5. Context Provider Average CPU Idleness as Load Increases

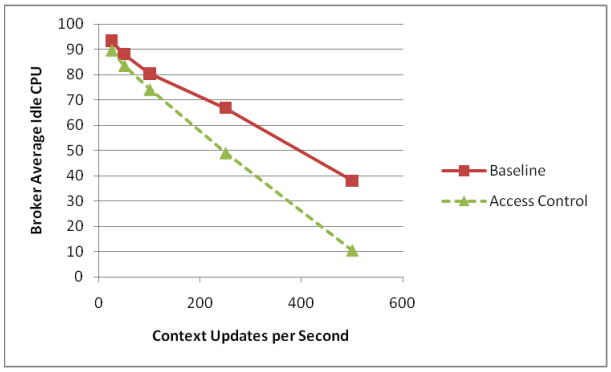

Fig. 6. Context Broker Average CPU Idleness as Load Increases

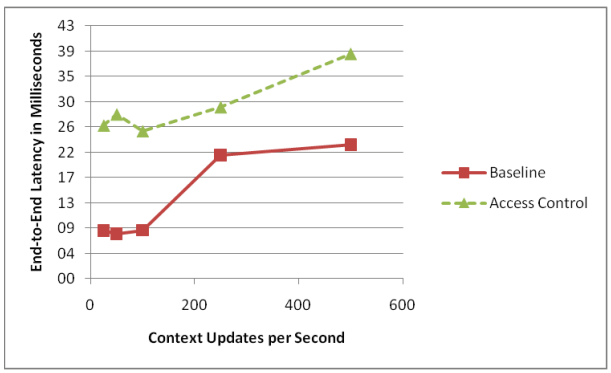

Fig. 7. End-to-End Context Delivery Latency with Load Increases

In each test session a number Context Sources published context at a fixed rate, and the Consumers got that content and wrote it with timestamps to a file. Furthermore, the CPU usage in both the Context Broker and Context Provider was measured, for both access control and baseline cases. The CPU measurements are based on the idle CPU output from vmstat [18]. The output shows average values for a sampling period that was set to 2 seconds. Since all the virtual machines only have one core, we don't have to worry about multi-core CPU measurement issues. The result gathering was repeated in 5 sessions to detect odd events. Correlation tests show that the same behaviour was observed in all test sessions,

with relative standard deviation well below 10% for all cases, except in Broker CPU measurements with 500 context updates per second on the Access Control case. In this case the relative standard deviation reaches almost 15%, due to the Broker not being able to handle all the load at times. In fact, the Broker processing load at around 500 contexts updates per second was the encountered bottleneck for the tested deployment. Figures 5, 6 and 7 show the relevant findings.

As we test with higher load, the average CPU occupation rises linearly, both for the Provider, as seen in Figure 5, and for the Broker , shown in Figure 6. In fact, in the Provider case, the average CPU occupation is practically the same in both implementations. The only difference is a slightly bigger memory footprint of the Access Control Provider. In the Broker, however, the linear increase in CPU demand for the access control case is clearly faster than for the original implementation.

To properly analyze the rise in latencies, we have removed the latency data referring to the first 2 minutes of each session, both for original and access control cases. The reasoning for this is that the system takes some ammount of time, in every case under 2 minutes, to stabilize its performance. We came to this value from the CPU measurements, which show substantially higher loads in the first seconds of each session. Furthermore, since this is meant to be a system that is always running, the relevant results are the ones we obtain after the startup. The results are depicted in Figure 7, which shows that context submitted to access control clearly takes more time to reach the destination, under any load. However this additional delay is estimated in around 20ms, a value perfectly in line within the human perception of "real-time", representing roughly 10% of the human visual reaction time of 190ms.

VI. Conclusion and Future Work

The authors presented a system capable of distributing context information in a controlled way that fulfills the real-time requirements of the process. The system considers fine-grained context privacy settings, allowing the user to set the context type and the valid parameters for which information should be published. Despite the binary result of an access control decision, allow or deny, complex effects could be implemented with parameter comparison functions other than regular expressions, which were chosen for this work. For example, a profile might be created that only provides the GPS location based on a probabilistic function centered on some coordinates. This would result on a "fade effect" of the location updates. Furthermore, considering status updates to be context information, a profile that only accepts status updates in English could be defined.

A bottleneck on the Context Broker was detected when load increases above certain values. However this is only a deployment problem which can be mitigated by decoupling the Publish-Subscribe Service from the XMPP Server. This shall be considered in the future by the authors, as additional privacy functionality is added. A key example of such functionality to be addressed in future work is the use of obfuscation by chaining context providers based on privacy settings, as considered in the current architecture. Additionally, the use of identity management techniques for allowing some degree of anonymity when publishing context information is also to be targetted.

Acknowledgment

The work presented in this paper was partially supported by the European Comission via the ICT FP7 SOCIETIES Integrated Project (No. 257493) [19].

References

[1] J. Tsai, P. Kelley, L. Cranor, and N. Sadeh, "Location-sharing technologies: Privacy risks and controls," *ISJLP*, vol. 6, pp. 119–317, 2010.

[2] M. Benisch, P. Kelley, N. Sadeh, and L. Cranor, "Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs," *Personal and Ubiquitous Computing*, pp. 1–16, 2010.

[3] N. Li and G. Chen, "Sharing location in online social networks," *IEEE Network*, vol. 24, no. 5, pp. 20–25, Sep. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5578914

[4] R. Kosinski, "A literature review on reaction time," *Clemson University*, vol. 10, 2008.

[5] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and capturing people's privacy policies in a mobile social networking application," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 401–412, 2009.

[6] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, vol. 16, no. 2, pp. 97–166, 2001.

[7] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[8] M. Zafar, N. Baker, B. Moltchanov, J. Gonçalves, S. Liaquat, and M. Knappmeyer, "Context Management Architecture for Future Internet Services," *ICT Mobile Summit 2009*, 2009.

[9] D. Gomes, J. a. Gonçalves, R. Santos, and R. L. Aguiar, "XMPP based Context Management Architecture," *Globecom 2010 - 4th IEEE Workshop on Enabling the Future Service-Oriented Internet: Towards Socially-Aware Networks (EFSOI 10)*, 2010.

[10] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," Tech. Rep. [Online]. Available: http://tools.ietf.org/html/rfc6120

[11] E. Toch, R. Ravichandran, L. Cranor, P. Drielsma, J. Hong, P. Kelley, N. Sadeh, and J. Tsai, "Analyzing use of privacy policy attributes in a location sharing application," *Proc. ACM SOUP*, 2009.

[12] C. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "Location privacy protection through obfuscation-based techniques," *Data and Applications Security XXI*, pp. 47–60, 2008.

[13] R. Wishart, K. Henricksen, and J. Indulska, "Context privacy and obfuscation supported by dynamic context source discovery and processing in a context management system," *Ubiquitous Intelligence and Computing*, pp. 929–940, 2007.

[14] T. Moses, A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, E. Coyne, F. Siebenlist, H. Lockhart, M. McIntosh, M. Kudo, P. Humenn, R. Jacobson, S. Proctor, S. Godik, and S. Anderson, "eXtensible 2 Access Control Markup Language (XACML) Version 2.0," Tech. Rep., 2004.

[15] IgniteRealtime, "Smack." [Online]. Available: http://www.igniterealtime.org/projects/smack/index.jsp

[16] Sun, "Sun XACML Implementation." [Online]. Available: http://sunxacml.sourceforge.net/

[17] IgniteRealtime, "Openfire." [Online]. Available: http://www.igniterealtime.org/projects/openfire/

[18] H. Ware and F. Frédérick, "vmstat." [Online]. Available: http://linux.die.net/man/8/vmstat

[19] "SOCIETIES." [Online]. Available: http://www.ict-societies.eu